热门文章 编辑推荐 原创

Messiah中GPU Culling实现笔记

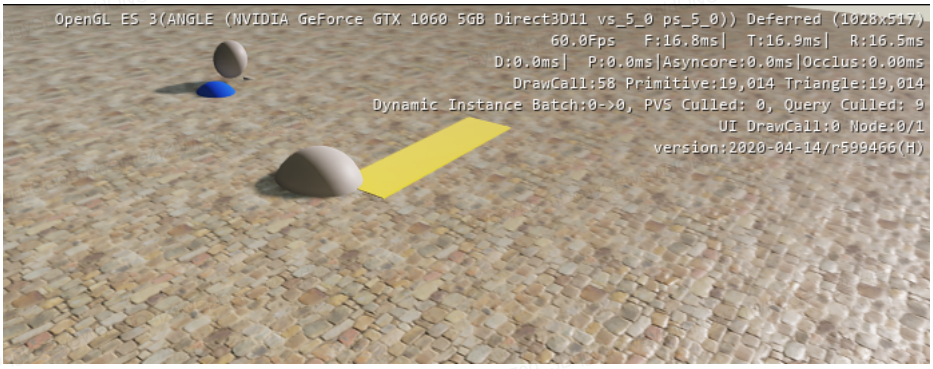
Kanglai(钱康来)

发布时间:2020.04.06 17:52 1363 30 1 更多

已推荐到：职业精选-程序/游戏客户端、今日看点-4.7

最近因为项目需要，在Messiah中实现两套GPU Culling实现：基于Query和基于HZB，并有了一定的初步测试结果。

最近由于项目需要尝试一些动态遮挡剔除方案。CPU版本(原理及实践可参考《实时软遮挡剔除方案》 写的很详尽) 我们测试了Messiah中自带的实现，(搞了下Proxy Geometry)但性价比不是特别理想，所以想尝试下GPU版本实现。



1 基于Query版本

2 思路原理

原理见KM上已经有很好介绍《UE4的硬件遮挡查询》、《G93遮挡剔除方案》，比较简单没什么好说的...主要记录一下实现过程中的踩坑地方，备忘一下。

3 整体流程

Messiah 2019.3版本已经提供了不少自定义管线相关接口，rdt层主要是注入了两个地方接入功能，还是比较方便的。

OcclusionQuery

主流程调用，直接塞进RenderPipeline里。这里我最早版本是放进RenderScene里，后来发现这货ping pong的访问方式导致错一帧无法获取到对应的Query...

具体调用就直接塞在RenderPipeline::_TickedViewport_on_rdt里，对于SceneCuller视锥剔除之后的prims再跑一轮GPU Culling即可：这个是核心实现函数，判断每个prim是否有历史可见信息，以及是否需要生成对应的包围盒去跑深度测试。

TODO: 目前有一个不太好做的事情是因为不同IPrimitive的CollectPrimitive函数里会有各种检查来跳过(譬如Reflection View下只渲染标记IsReflectionVisible的东西)，但这个没有统一接口暴露所以只能一口气全塞进GPU Culling里去。

OcclusionTest

自定义RenderPass，作用是修改SceneCuller的TargetSet带上ERenderSet::OcclusionTest，然后绘制所有的Occludee即可。

这里需要注意的就是Depth不要触发resolve直接move(对于非downsample情况)，避免出现rt的save/load。

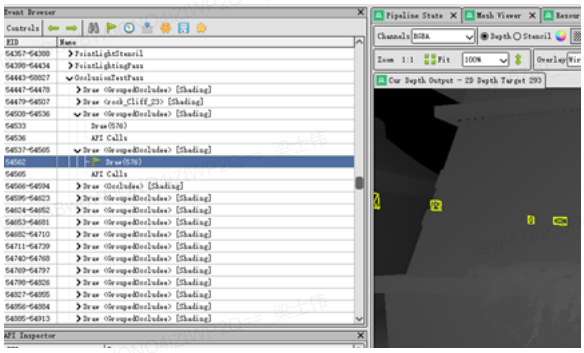
rdt层主要就是OcclusionQuery里: 对于需要跑GPU Culling的物体，获取世界空间包围盒之后生成一个Cube，然后塞进RenderElementOccludee之后调用scene->_AddPrimitive_on_rdt就行了；同时在返回的RenderItem上传一下RenderQuery，这样一路传到CommandEncoder的IM_Draw部分就打通了正向逻辑。
比较麻烦的是反向逻辑：如何从dt层获取数据并传回rdt，因为messiah最多可能出现3帧错位，所以我目前的实现是参考EventCallbackOperation搞了个EventQueryCallbackOperation，从dt层直接回调过去，然后暴露IQueryCallbackContext封装GetData从而使数据流向正常。

5 实现细节

在DX11上第一版实现完了之后直接发现性能血崩...看了眼是DrawCall瓶颈，打开GPU Culling之后直接从2k翻倍到4k了。所以实现细节主要围绕如何减少每帧Query带来的额外DrawCall上（在保持保守策略前提下）

Group Query

最简单的策略其实就是多个物体一起查询，所以可以把上一帧确认不可见同时本身不是Movable的Entity归并到一起去查询，但每次归并的也不能太多因为只要其中一个可见就只能认为都可见了。



避免单帧频繁查询

也是类似的情况：对于动态物体只能每帧查询，但是静态物体不需要这么频繁——如果上一帧没有被挡住，而且通过深度测试的像素越多，下一帧依然没有被挡住的概率就越大，因此需要查询的周期可以拉长（保守策略标记为可见）

DepthDownSample

这里感谢蔡泽野老哥指导~ 我本来是准备直接接入艺神实现的HierarchyZ，但是发现带mip的R16很难各个平台resolve成Depth直接使用(DX11没问题，ES3上搜了下貌似不行)。我试了下硬Copy一次高mip结果到另一个新的ShadowDepth倒是也行，但略显奇怪。
G93是直接glBlitFramebuffer一个低分辨率的Depth，没经过HiZ但是效果没问题。但这样的局限是只支持GL，我没找到DX11/Vulkan/Metal下对应API（提供的都是硬copy，没有scaling）
我最后还是用一次全屏Draw解决问题，Shader里直接去写SV_Depth测试兼容性也刚刚的。

6 RHI部分

TransientBuffer

由于每帧都需要动态拼装RenerElementOccludee，所以使用TB一口气去更新buffer，同时偷懒这里也直接使用VB硬画、不需要IB。

EventQueryCallbackOperation

参考EventCallbackOperation实现，主要是目前FrameGraph这块的Callback实现都是不全的，需要人肉补全一下(EDeviceMultiThreadLevel_Solo和EDeviceMultiThreadLevel_Cooperative都得搞下)

DX11

DX11的API封装是最简单的，从对象池里获取ID3D11Query然后Begin/End/GetData即可。因为是错帧，所以GetData的时候直接加个while循环等到结果返回即可。
GetData还遇到过一个有意思的bug：我加了CheckHybrid检查GetData返回值只可能是S_OK或者S_FALSE(表示数据未就位)，但是有时候出现DXGI_ERROR_INVALID_CALL，用D3D Debug Layer看也没有任何有效信息。最后发现是绘制Occludee的时候Shader Warmup会导致Query根本没Issue...

Vulkan

Vulkan相比前两个平台开始麻烦起来了，因为Vulkan的Query天然就是一个Pool，不能单个使用。我目前的做法是类似QcclusionQuery的Callback一样，先预先创建多个QueryPool，然后每帧轮换使用不同的QueryPool。
为了和之前DX11/ES3的接口保持一致，VulkanQuery搞成了一个单纯逻辑的包装，对应VulkanQueryPool及Index(代表Pool里第几个可用Query)来使用。
目前有个实现的比较丑的地方是获取数据的时候是一个个VulkanQuery单独做的，本来想一口气把VulkanQueryPool的数据都回读，但是如果遇到因为Shader Warmup导致有些Query没有Issue的话，这么做根据文档会出现卡死...

Metal

Metal的实现和Vulkan其实很像，在创建renderpass的时候把visibilityResultBuffer喂进去，然后就可以使用了。所以MetalQuery也是一个单纯逻辑包装，对应MetalQueryPool及Index。然后MetalQueryPool里对应的id<MTLBuffer>用来读写数据。
在实现过程中Metal是我改的最绕的，因为Messiah会走MergePass逻辑，也就是可能出现多个RenderPass会合到一起，这时候如果在遇到OcclusionTest的时候才设置visibilityResultBuffer就已经晚了。所以我在BeginMainPass的里面先搜一遍所有MergedPass是否需要设置，然后提前到BeginRenderPass时就能设好。

HZB版本

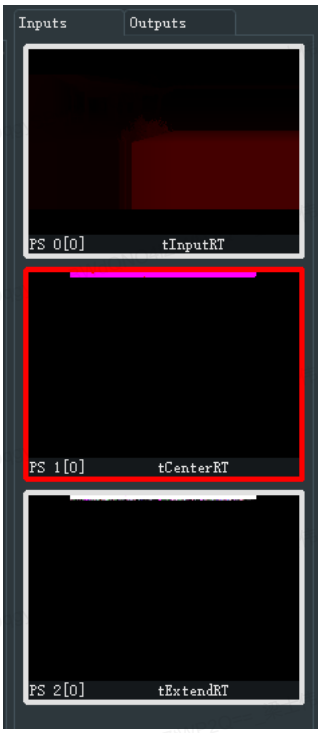
7 思路原理

后来在原来的框架基础上抽空做了HZB版本。原理很简单：在Shader里计算包围盒在HierarchyZ的DepthMap里有没有被遮挡。为了兼容性目前是用Pixel Shader模拟的，其实后面用Compute Shader更好。
和Query版本的实现相比

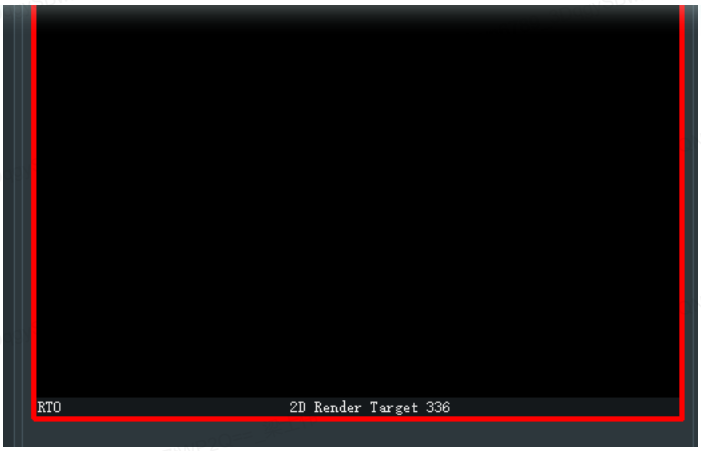
- 优点：一次Draw就能解决问题，所以之前的GroupQuery策略之类的都不需要了
- 缺点：每帧需要CPU Write和CPU Read...而且由于策略更加保守，所以能剔除掉的Primitive会少一点

8 整体流程

关于HiZ的DepthMap可以参考Hierarchical Depth Buffers，艺神已经做了一个版本我直接拿来用了。
输入三张贴图：HiZ版本的DepthMap，以及两张float rt分别存储了包围盒的Center和Extend



输出一个RGBA8888的RT，存储是否可见（白色像素）

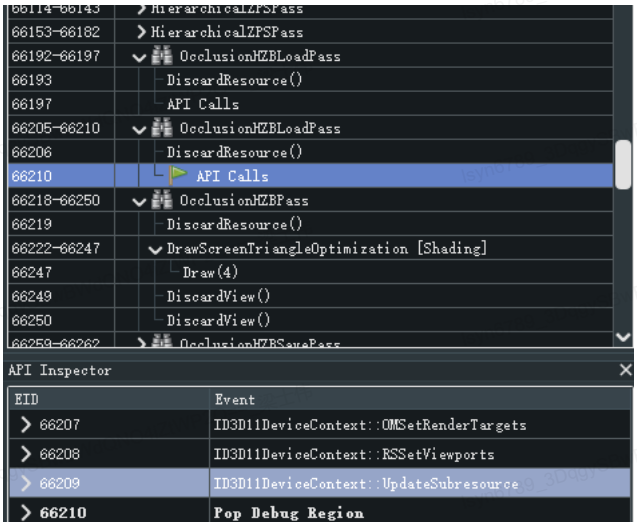


实现细节

实现起来在原来框架上改改就行了，就两个细节值得提一下

1. 因为是模拟Compute Shader，所以贴图采样应该是Point，避免插值带来的问题
2. 为了结合FrameGraph自动处理RT复用，所以这里的tCenterRT/tExtendRT以及Output全都是ERenderTargetLive_Transient的。

结果的CPU Read比较简单，FrameGraph已经暴露了Save操作；为了劫持进CPU Write，我额外加了一个Load操作进去



题外话：目前自带的Save(GPU->CPU)、曹哥哥加的Copy(GPU->GPU)、我刚加的Load(CPU->GPU)三个操作算是能满足目前所有需求了

RHI细节之ES3

多平台版本实现过程中唯一遇到问题的反而是ES3，当时发现HiZ的mip始终写入不能，所以DepthMap一直是错的。然后灵机一动去翻了翻别人SDK版本里的实现hizculling.cpp

```
// Need to do this to ensure that we cannot possibly read from the miplevel we are rendering to.
// Otherwise, we have undefined behavior.
GL_CHECK(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, lod - 1));
GL_CHECK(glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_LEVEL, lod - 1));
```

对着这个发现messiah这部分的功能目前注掉了，而且Sub部分的语义有歧义...动手修了下这里就好了。

真机测试

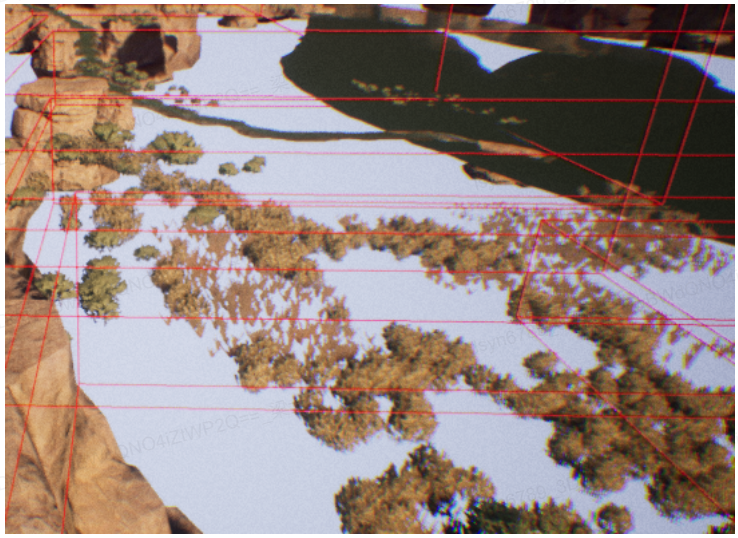
目前只是在新项目简单测试了一下：

- Query版本：野外场景iPhone8上从38fps涨到41，小米9上从40fps涨到45，有一定效果但是还有不小优化空间。PC 编辑器上效果不是很明显(DX11从33fps涨到36)主要是视距太远，原因是剔除之后DrawCall依然爆炸。
- HZB版本：也就PC上开这个有一定意义（能多提升几帧效率，毕竟卡DrawCall）；iPhoneX上测试了下HZB还不如不遮挡剔除...

所以手机上目前来看还是基于Query版本更佳。

ps. 后来正好 【易学堂】第88期：高通骁龙游戏优化主题分享 - GPU 篇 里高通的人也提到这块相关内容，其中GPU Culling部分的结果我和测下来的趋势是一致的。准备有空跟进下他们特化版的软光栅，希望能有惊喜。

是能够特化处理的。UE4里HISM直接抽取上层节点的包围盒使用，因此需要等钊哥实现一下之后重新接入。



- 如果继续往GPU Driven方向逼近的话，即用Compute Shader版本实现然后直接GPU组装Draw Command的话，从ExecuteIndirect investigation来看(感谢曹哥哥分享的链接) DX和Metal都是API Ready的状态，值得尝试... 不过这样需要进一波剥离GPU Scene数据了，配合mesh cluster之类的就是从资源管线动刀子了...~~天坑警告~~

全部评论 1

请输入评论内容

还可以输入 500 个字

(可添加1个视频+5张图片) 匿名 评 论

最热 最新

夜川(周顺)

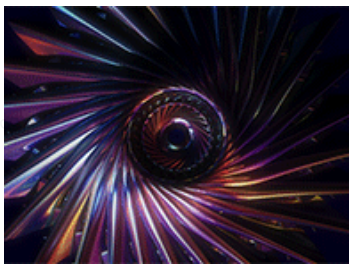
1楼 沙发膜拜

2020-04-20 14:40 回复 0

Kanglai(钱康来) 顺爷莫黑...我只是在引擎现有框架下做点微小工作

2020-04-20 14:51 作者回复

大家都在看



RenderDoc接入及使用技巧

UE4进阶培训简介

UE5 Nanite 浅析 (一) : 核心思路